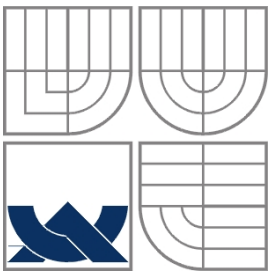
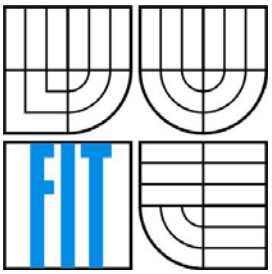


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZOBRAZENÍ ŠACHŮ POMOCÍ SLEDOVÁNÍ PAPRSKU

RENDERING CHESS USING RAY TRACING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. VOJTĚCH TUREK

VEDOUCÍ PRÁCE
SUPERVISOR

ING. ADAM HEROUT, PH.D.

BRNO 2008

Abstrakt

Práce se zabývá technikou zobrazování pomocí sledování paprsku, zaměřenou na vykreslování šachové scény a popisu materiálů pomocí procedurálních textur. Na základě těchto poznatků je vytvořen program, který zobrazí zadaný stav šachové partie.

Klíčová slova

Sledování paprsku, procedurální textury, šachy, phong, CSG, rotační plochy, subdivision, supersampling, perlinův šum, C++, implementace

Abstract

This work deals with ray tracing technique, focused to rendering of chess scene with procedural texture materials. Based on this knowledge, program that displays specified chess set is designed.

Keywords

Ray tracing, procedural textures, chess, phong, CSG, sweep surfaces, subdivision, supersampling, perlin noise, C++, implementation

Citace

Turek, Vojtěch : Zobrazení šachů pomocí sledování paprsku.
Brno, 2008, diplomová práce, FIT VUT v Brně.

Zobrazení šachů pomocí sledování paprsku

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením p. Ing. Adama Herouta, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jméno Příjmení
15.5.2008

Poděkování

Chtěl bych na tomto místě poděkovat Ing. Adamu Heroutovi, Ph.D. za odborné vedení a konzultace, které mi poskytl při práci na této diplomové práci.

Obsah

1 Úvod

Pod obor počítačová grafika můžeme zahrnout všechny metody a techniky, které při vytváření vizuálního vjemu využívají počitače. Nejvíce rozšířené a používané jsou ty, které pro svůj výstup používají dvojrozměrný prostor – monitor. V současnosti se dají zhruba rozdělit do tří oborů:

- Rasterizace je založena na popisu objektů pomocí řady primitiv, které určují plášť objektu. Tyto primitiva lze poté vykreslovat přímo grafickými akcelerátory. Je to nejrychlejší metoda a proto se používá v běžné technické praxi k vizualizaci a virtuální realitě.

- Technika sledování paprsku rozšiřuje možnost popisu objektů o objemovou reprezentaci. Lze i přímo zobrazovat plochy popsané algebraickými vzorci, např kvadriky apod. Je založena na paprskové optice. Z pomyslného oka se vyšle paprsek který protíná jednotlivé pixely průmětny výsledného 2D obrazu a výsledná barva se určí na základě průsečíků s objekty scény. Díky objemové a matematicky korektní definici objektů se technika sledování paprsku používá i v jiných technických odvětvích, jako například simulace šíření magnetických vln apod. Je řádově pomalejší jak rasterizace, umožňuje však věrnější zobrazení, zejména odrazů a stínů. Využívá se pro generování kvalitních, přesných a fotorealistických vizualizací.

- Globální osvětlení nejlépe vystihuje fyzikální realitu. Sledují se paprsky energie vyzářené zdroji. Tato metoda je v současné době nejvíce realistická (v rámci možností). Cenou je však řádově delší výpočet než u metody sledování paprsku.

Sledování paprsku představuje dobrý kompromis mezi kvalitou a rychlostí výpočtu. Vzhledem k dostatku materiálů zabývajících se touto technikou ji lze i poměrně snadno implementovat.

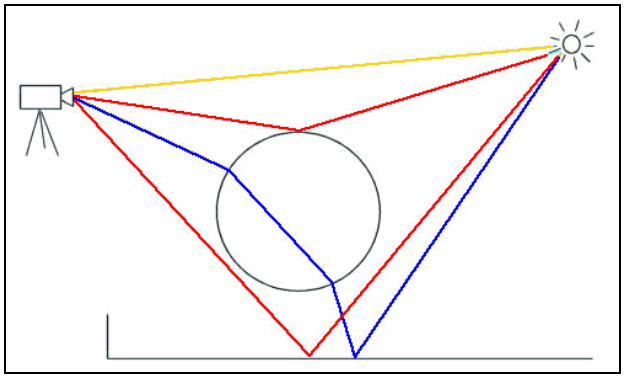
Předmětem této práce je vytvořit program, který dokáže zobrazit šachovou scénu. Nejdříve tedy bude popsán teoretický základ v potřebném rozsahu. Poté navržen způsob popisu objektů a jejich materiálů a vybrány metody a algoritmy pro popis prostředí. A nakonec návrh algoritmů a implementace programu.

Diplomová práce navazuje na semestrální projekt, jehož obsahem bylo nastudování problematiky a návrh, který je v kapitolách 2 a 3.

2 Metoda sledování paprsku

2.1 Obecné principy

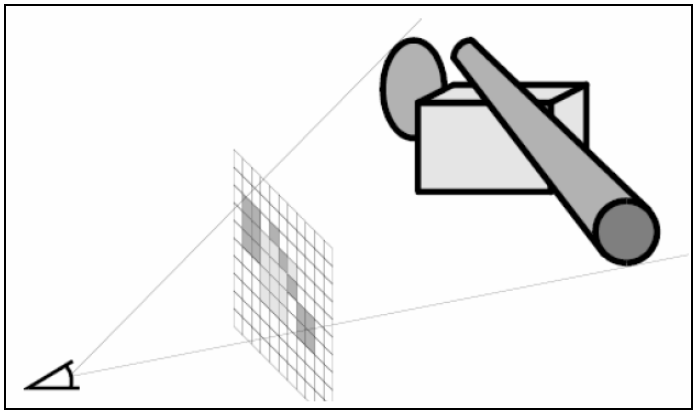
Je to v podstatě pokus napodobit šíření světla v přírodě. Barvy, které vidíme, jsou paprsky energie vržené sluncem či jiným zdrojem světla. Ty se odrážejí od krajiny a nakonec skončí na naší sítnici, kde je vnímáme jako obraz. Pokud zanedbáme vlivy prostředí, jako magnetismus, teplota vzduchu apod., můžeme paprsky reprezentovat polopřímkou. Při dopadu světla na povrch materiálu mohou nastat různé kombinace tří jevů: absorpce, odraz a lom.



Obrázek 2.1

Jak ukazuje obrázek 2.1, žlutý paprsek jde přímo ze slunce do kamery. Červený do kamery vstoupí až po několika odrazech. Modrý paprsek láme skleněná koule. V obrázku však chybí paprsky, které nikdy nevstoupí do kamery. A to je myšlenka raytracingu. Protože z geometrického pohledu nezávisí na směru paprsku, můžeme počítat pouze s paprsky které vycházejí z kamery.

Pro každý pixel výstupního obrázku (obr. 2.2) se vyšle primární paprsek ve směru oko-pixel v průmětně scény. Zjistí se nejbližší průsečík s objektem a určí barva na základě normály objektu a sekundárních paprsků. Ty jsou vyslány ke světlu (stínový paprsek), a v případě odlesků, odrazů a lomu světla, dle fyzikálních zákonů dále rekurzivně do scény.



Obrázek 2.2, převzato z [5]

Sledování paprsku se používá jak v aplikacích, kde je třeba matematicky přesného modelu zobrazení, tak i při návrhu optických systémů jako jsou fotoaparáty, mikroskopy a podobně. Mezi výhody patří:

- realistické osvětlení a stíny
- odrazy a lom světla které je obtížné simulovat u běžných metod
- jednoduchost implementace
- možnost širšího výběru matematického popisu objektů než z rasterizace (csg, implicitní plochy, kvadriky, fraktály apod)
- jednoduchá paralelizace výpočtu

Má také ale dost nevýhod:

- pomalost výpočtu – převážně se nehodí pro realtime použití
- pořád ještě není dostatečně realistická – ostré stíny, osvětlení neodpovídá světelné rovnici apod.
- absence hardwarové podpory

2.2 Popis objektů

Protože se nejedná o realtime metodu, můžeme objekty popsat libovolnou matematickou metodou – jediný požadavek je, aby jsme byli schopni vypočítat průsečík s paprskem a určit normálu v bodě.

- Grafická primitiva jako koule, kvádr, trojúhelník, válec apod. jsou nejjednodušší modely co můžeme použít. Výpočty průsečíku vycházejí z parametrického vyjádření paprsku a obecných rovnic těles.
- CSG (constructive solid geometry) modely jsou takové modely, které vznikly z grafických primitiv pomocí tří booleovských operací: $\cup$ (sjednocení), $\cap$ (průnik), $-$ (rozdíl). Operace se skládají do binárních stromů. Spolu s grafickými primitivy jsou nejběžnějšími metodami pro popis objektů.
- Implicitní plochy umožňují zobrazit matematické funkce či různě zadaná medicínská data. Znamená to řešit rovnici f(x,y,z)<=konstanta. Pokud je rovnice splněna, pak se bod nachází uvnitř objektu. Nejčastěji bývá funkce součtem dílčích funkcí, které „vyzařují energii“ v závislosti na poloze. Spolu s NURBS plochami tvoří hlavní možnosti reprezentace „nehranatých“ těles.
- NURBS (non uniform rational b-splines) značí plochy vytvořené za pomocí b-splinů.

2.3 Materiály a osvětlovací modely

2.3.1 Osvětlovací model

Při dopadu paprsku na těleso se světlo může pohltit či odrazit, změnit směr či vlnovou délku apod. Je třeba řešit tzv. obrazovou rovnici, která staví do rovnice energii vyzařující z objektu a součet energií odraženého světla a vyzařované energie objektem. Výsledná energie určuje barvu.

$$L_0(x,\vec{w}) = L_e(x,\vec{w}) + \int_\Omega f_r(x,\vec{w}',\vec{w}) L_i(x,\vec{w}')(\vec{w}' \bullet \vec{n}) d\vec{w}' \text{ , kde}$$

$L_0(x,\vec{w})$ je celková výstupní energie objektu

$L_e(x,\vec{w})$ energie vyzařující objektem

$L_i(x,\vec{w}')$ je světlo zářící z daného směru

$(\vec{w}' \bullet \vec{n})$ je útlum světla vzhledem k úhlu světla k normále povrchu

a $f_r(x,\vec{w}',\vec{w})$ je BRDF (bidirectional reflectance distribution function). Funkce určí intenzitu světla v závislosti na vektoru dopadajícího paprsku a dopadajícího světla. Existuje mnoho způsobů jak výpočitat tuto funkci. Jednou z nejstarších je Phongův osvětlovací model. Lze snadno implementovat a má přijatelné výsledky (viz 2.2). Přestože bylo vymyšleno mnoho realističtějších metod, jako Torrance and sparrow, Schlickův model a další, přesto se jedná o nejběžněji používanou metodu.

$$I_p = k_a i_a + \sum_{lights} (k_d (\vec{L} \bullet \vec{N}) i_d + k_s (\vec{R} \bullet \vec{V})^\alpha i_s)., \text{ kde}$$

k_a je koeficient intezity všesměrového světla, které má stejnou intenzitu ve všech bodech scény

k_d je koeficient odrazu difuzního světla od objektu.

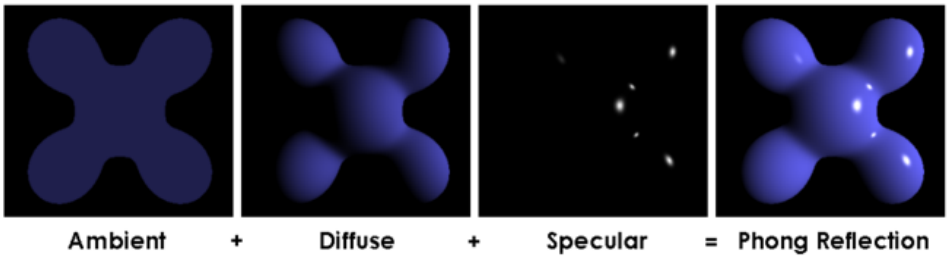
k_s je koeficient lesklosti povrchu.

L je vektor ve směru od průsečíku s objektem ke světlu

R je vektor L odražený od povrchu tělesa

V je vektor dopadajícího paprsku

N je normála povrchu



Obrázek 2.3, převzato z [4]

Difuzní a spekulární složka se nezapočítává, pokud bod leží ve stínu, tj. stínovému vektoru z průsečíku ke světlu leží v cestě nějaký jiný objekt. Vzorce a výpočet byl převzat z [4].

2.3.2 Odraz a lom světla

Na rozdíl od metody raycasting, kde se v průsečících počítá pouze osvětlení, ray tracing v případě reflexního či průhledného povrchu rekurzivně vyšle odražený, resp. lomený paprsek. Odražený paprsek se spočítá $\vec{V}' = \vec{V} - 2 * \vec{N}(\vec{V} * \vec{N})$, kde V je vektor dopadající a N normálový vektor.

Lom světla se řídí snellovým zákonem:

$$\vec{t} = \frac{\eta_1}{\eta_2}\vec{i} - (\frac{\eta_1}{\eta_2}\cos\theta_i + \sqrt{1 - \sin^2\theta_t})\vec{n}$$, kde

η_1, η_2 jsou indexy odrazu

$$\cos\theta_i = \vec{i} \bullet \vec{n} \quad , \quad \sin^2\theta_t = \left(\frac{\eta_1}{\eta_2}\right)^2 \left(1 - \cos^2\theta_i\right)$$

Výsledné intenzity světla se poté přičtou k výsledku osvětlovacího modelu. Převzato z [2].

2.4 Texturování

Přidáním textury můžeme dodat povrchu objektu detail, aniž by jsme ho museli geometricky modelovat. Znamená to, že se při výpočtu barvy v místě dopadu paprsku nepoužije konstantní materiál pro celý objekt, ale určí se pomocí funkce která danému bodu v prostoru přiřadí určitý bod v prostoru textury. Pomocí texturovaní lze modifikovat jak konstanty phongova osvětlovacího modelu, tak i směr normály nebo dokonce i definovat daný bod jako průhledný. Pomocí této techniky lze dodat scéně poměrně snadno realistický vzhled.

2.4.1 2D Texturování

Nejběžnější technikou je namapování 2D bitmapy na 3D povrch. Znamená to, že se při výpočtu osvětlení nahradí difuzní barva materiálu vzorkem z bitmapy. Průsečík s objektem se přepočítá na lokální souřadnice objektu tak, aby bylo možné je transformovat do 2D prostoru textury. V případě koule na sférické souřadnice, u válce cylindrické souřadnice, u trojúhelníku na kartézské apod.

2.4.2 Procedurální texturování

V procedurálním texturování se materiál určí výpočtem funkce c=PT(p), kde p je průsečík s objektem přepočtený na lokální souřadnice. Funkce PT je algoritmus nebo vzorec, který vypočítá potřebné materiálové vlastnosti. V procedurálních texturách zpravidla není třeba převodu prostorů jako u 2D texturování, lze generovat přímo v 3D. Pro generování je k dispozici celý matematický aparát. Lze začít jednoduchými funkcemi pro generování pruhů (např. PT()=sin(p.x)>0?1:0), dále pokračovat přes kombinace jednoduchých periodických funkcí, ke křivkám apod. Vlastní skupinou jsou šumové

funkce, které slouží jako základ pro tvorbu většiny materiálů. Jednou z nejrozšířenějších je perlinův šum.

Výhodou procedurálních textur je jejich zanedbatelná náročnost na paměť, v porovnání s bitmapovými texturami. Většinou jsou spojité, odpadá tedy problém filtrace diskrétních hodnot. Nejsou problémy s opakováním a navazováním textur. Jsou lehce parametrizovatelné, a na rozdíl od bitmapových tvoří celou třídu materiálů.

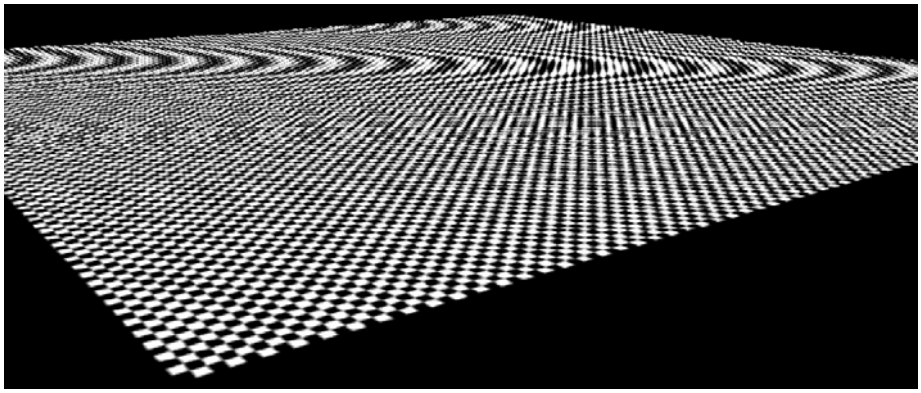
Mezi jejich nevýhody však patří delší doba výpočtu a náročnější tvorba (výsledný efekt si při programovaní stěží dokážeme představit).

2.4.2.1 Perlinův šum

Ken Perlin vynalezl šumovou funkci, která má ideální vlastnosti pro použití v počítačové grafice. Lze ji rychle vypočítat a splňuje potřebné požadavky: je statisticky invariantní vzhledem k otáčení a posunutí, je spojitá, má omezené frekvenční spektrum a je opakovatelná. Pro stejné parametry vrátí vždy stejný výsledek. Srdcem metody je součet stejné šumové funkce o různé intenzitě a v různé frekvenci. Výsledný šum může být podkladem pro třídu textur pro mramor, dřevo, mraky a další.

2.4.3 Aliasing při mapování 2D textur

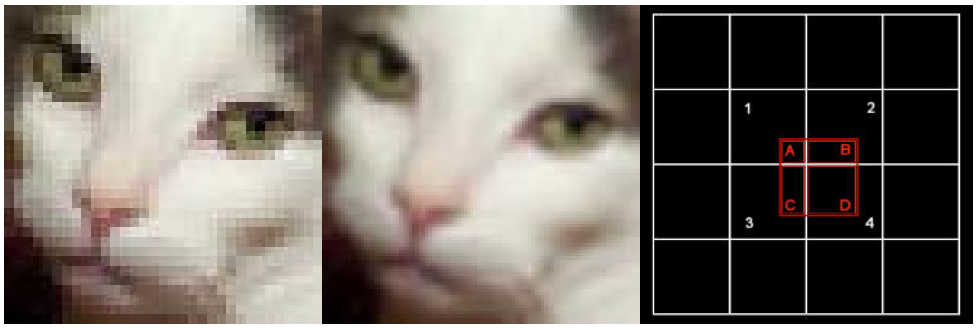
Mapování 2D textury je vztah mezi diskrétním obrazem textury a obrazovky. Při mapování v závislosti na vzdálenosti se body textury zvětšují, příp. zmenšují. Dochází tím k nerovnoměrnému vzorkování obrazu, odborně nazváno, aliasingu. Aby k němu nedocházelo, jeden pixel by se musel mapovat vždy na jeden texel. Musíme tedy buď zvýšit vzorkovací frekvenci pixelu nebo snížit maximální frekvenci v textuře.



Obrázek 2.4

Obrázek 2-7 ukazuje nepříjemné efekty aliasingu. Existuje několik metod jak se s tím vypořádat.

První je filtrace. Při vybírání texelu z textury lze použít interpolaci. Bilineární, trilineární, anizotropickou aj. Vybraný fragment bude mít barvu určenou váhovým součtem barev okolních pixelů, v závislosti jakou vahou pozice zasahuje do ostatních pozic texelů (viz obr. 2.5). Nicméně si tato metoda nedokáže poradit s vysokými frekvencemi.



Obrázek 2.5

Další metodou je MIP mapping. Zkratka znamená „multimum in parvo“ – tedy „mnoho v malém“. Uchovává se několik textur v různé frekvenci. Při vykreslování se vybere ta, která bude nejblíž vzorkovací frekvenci obrazu. Jednotlivé přechody mezi frekvencemi interpolují pomocí blendingu. Na obr. 2.6 lze vidět uložení MIP textur v paměti.

Obrázek 2.6, převzato z [5]

Tato metoda si nedokáže poradit s nízkými frekvencemi obrazu. Respektive teoreticky ano, ale z důvodu šetření pamětí a zpravidla omezenou velikostí textury ne. Obvykle se tedy filtrace a MIP mapping kombinují dohromady.

2.4.4 Supersampling

Z důvodu spojitého charakteru výpočetní funkce procedurálních textur nenastává aliasing při nízkých frekvencích. Při vysokých ale nastává problém. U procedurálních textur je filtrace obtížná, nebo dokonce i nemožná.

Řešením je supersampling. Místo jednoho paprsku na pixel se jich počítá více, vhodně rozmístěných. Výsledná hodnota pixelu je aritmetickým průměrem získaných hodnot. Je to metoda velmi náročná na výpočet, neboť její časová složitost při výpočtu n-paprsků/pixel je stejná jako při výpočtu n krát vyššího rozlišení. Výsledek je však kvalitní.

Pro odstranění aliasingu na hranách objektů lze algoritmus zoptimalizovat. Adaptivní supersampling vyšle více paprsků jenom v případě, že se průsečíky s vrženým paprskem liší od předchozích. Výsledky jsou podobné s vyjímkou vyšší rychlosti. Vyjímkou jsou však texturované

objekty, pro které je třeba použít jiný druh vyhlazování nebo upravit raytracer tak aby je antialiasoval též.

2.5 Optimalizace

Sledování paprsku je obecně metoda velmi náročná na dobu výpočtu. Obzvlášť pokud implementace není urychlena. Úzké hrdlo aplikace je v množství průsečíků, které se musí při každém výpočtu paprsku zkontrolovat. Optimalizací samotného výpočtu průsečíků příliš nepomůže. Nezbývá než snížít počet objektů, se kterými se počítá při vržení paprsku.

2.5.1 Hiearchické obalové objekty

Pokud skupinu objektů, které leží blízko sebe, seskupíme a obalíme primitivním tělesem, pak můžeme, v případě že paprsek nemá průsečík s tímto obalovým tělesem, vynechat všechny vnořené objekty z výpočtu. Podobně lze seskupovat i obalové objekty a vytvářet tak stromovou strukturu. Touto metodou můžeme dosáhnout více jak desetinásobného zrychlení. Obalové objekty je však zpravidla nutné zadat ručně, což je obtížné u dříve vytvořených scén. Jako obalové objekty se většinou používají kvádry nebo koule. Užitečné je také uvědomit si, že při zjišťování kolize s obalovými tělesy nemusíme znát průsečík. Vystačíme si s informací že existuje. To může výpočet některých těles citelně zrychlit.

2.5.2 Metody dělení prostoru

Nejjednodušší metodou je dělení uniformní mřížkou. V každé buňce mřížky se uloží odkazy na objekty které leží v daném prostoru buňky. Při vrhání paprsku se tak počítají průsečíky pouze s objekty které leží v blízkosti paprsku. Je jednoduché ji implementovat. Funguje dobře ale jenom pro rovnoměrně rozmístěné scény. V opačném případě mrhají pamětí a v horším případě bude scéna umístěna v několika málo buňkách. Existuje spousta dalších metod dělení prostoru, jako octree, BSP tree apod. Z ostatních metod se zaměřím už jenom na adaptivní k-D tree, která má relativně dobrou charakteristiku.

k-D tree funguje podobně jako BSP tree, s tím rozdílem že dělení prostoru se provádí ve směru os. Scéna se prostorově rozdělí na dvě části, které mají přibližně stejné množství objektů. Obě části se opět rozdělí na dvě části. Takto se pokračuje rekurzivně, dokud v části nezůstane méně než určených objektů. Při výpočtu se prochází stromem na základě pozice paprsku.

3 Návrh programu

Než přistoupíme k návrhu programu, je třeba stanovit si požadavky a použité algoritmy. Cílem je zobrazit šachovnici s předem zadanou pozicí figurek. Program by měl využít možností raytracingu a procedurálních textur.

3.1 Kamera

Pro lepší parametrizovatelnost vizualizace je zavedena kamera, skrze kterou budeme scénu vykreslovat. Kamera má daný počátek, směr pohledu, zorný úhel a případně další parametry, jako jsou náklon v ose pohledu a poměr stran. Pro snadnější orientaci má kamera vlastní prostor. Ten je definován v jejím počátku (pozici), kde osy *x* a *y* jsou kolmé na směr pohledu. V tomto prostoru se vypočítá paprsek tak, že se k diskrétním souřadnicím pixelu dopočítá složka *z* vektoru paprsku, na základě úhlu pohledu. Poté se vzniklý vektor normalizuje a převede do prostoru scény:

```
Ray getray(pixel) {
  horizontalfov=((FOV/180)*PI)/2;
  ray.dx=pixel.x - ScreenWidth/2;
  ray.dy=pixel.y - ScreenHeight/2;
  ray.dz=(ScreenWidth/2)/tan(horizontalfov/2);
  return CameraToWorld(ray);
}
```

Kamera má funkce pro nastavení pozice a bodu pohledu. Z těchto souřadnic lze poté vypočítat vektor pohledu.

3.2 Prostory a konverze souřadnic

Pro snadnější orientaci v souřadnicích je výhodné definovat rozdílné prostory pro jednotlivé objekty scény. Již byl uveden samostatný prostor pro kameru. Dále existuje hlavní prostor, tzv. world space, ke kterému se vážou všechny pozice, ať už kamery, světel, modelů a podobně. Podobně je definován i samostatný prostor pro modely. Tento model se poté sestává z jednoho nebo více objektů, jejichž pozice a rozměry jsou určeny vzhledem k modelovému prostoru. Tímto rozdělením získáme možnost manipulovat s modely ve scéně, aniž bychom museli měnit definice geometrie objektů v nich uložených. Posledním prostorem je prostor objektů, který je definován vzhledem k prostoru modelu. Konverze mezi jednotlivými prostory se provádějí homogenními souřadnicemi pomocí transformační matice:

$$[x', y', z', 1] = [x, y, z, 1] \bullet \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ a_{41} & a_{42} & a_{43} & 1 \end{bmatrix}$$

Transformační matice vznikne složením (postupným vynásobením) dílčích transformačních matic, jmenovitě v našem případě pro rotaci ve třech osách , změně měřítka a posunutí.

$$Mx = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, My = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, Mz = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

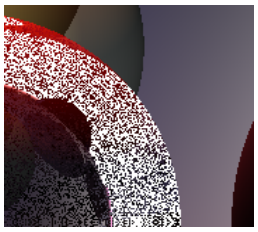
$$Ms = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, Mt = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{bmatrix}$$

Vzniklou matici můžeme použít i pro zpětný převod. Je však třeba vypočítat jeji inverzi. Tvar transformačních matic byl převzat z [6].

3.3 Sledování paprsku

Před pozorovatelem je umístěna průmětna, sestavená z pole pixelů, neboli výstupní obrázek. Těmito pixely jsou pak od pozorovatele vrhány paprsky výpočtem uvedeným v kap. 3.1. Poté se hledají průsečíky s objekty scény. Vyhodnotí se všechny možné, a vyberou ty, které jsou nejblíže k pozorovateli. Dále jsou vrženy sekundární paprsky. To jsou tzv. stínové paprsky, ke každému světlu je vyslán jeden. Pokud ve vzdálenosti od průsečíku ke světlu nalezne další průsečík, považuje se těleso ve stínu (pro dané světlo). Mohou být rekurzivně vyslány i další sekundární paprsky, v případě odrazivého nebo průhledného povrchu. Rozhodl jsem se použít empirický Phongův osvětlovací model, vysvětlený v kap. 2.3.1. Je jednoduchý na implementaci a výsledky jsou přijatelné.

Při vyslání sekundárních paprsků je třeba počítat s přesností matematické reprezentace čísel v počítači. Pokud se pro počátek paprsku použije přímo vypočtený průsečík, potom může dojít k tzv. akné. V závislosti na hodnotě čísla může najít průsečík s tím samým tělesem v témže místě. Výsledkem může jev na obr. 3.1.



Obrázek 3.1

Tento problém lze vyřešit tak, že se počátek paprsku posune ve směru sekundárního paprsku o velmi malou hodnotu.

Světla jsou modelována jako bodová, bez útlumu energie a s všesměrovým šířením. Jejími parametry jsou pozice a barva. Mají nulovou velikost a proto je nelze vidět.

3.4 Antialiasing

V kapitole 2.4.3 byla zmíněna problematika aliasingu. Ten může ve scéně nastat při mapování textur hodin, šachovnice nebo dřeva. Antialiasing dřeva by bylo možné odstranit MIP mapováním a filtrací textury. Ve scéně však zabírá pouze malé místo tudíž pro jeho implementaci to není dostatečná motivace. Dále alias nastává při mapování textury šachovnice a dřeva při nízkých rozlišeních. Zde by bylo odstranění složité, proto jsem se rozhodl nepodporovat filtrování či MIP-mapování, bude implementována metoda supersamplingu, kterou bude možné odstranit alias vysokých frekvencí.

3.5 Optimalizace

Zde se jednoznačně nabízí metoda obalových těles. Figurky jsou jedinými komplexními objekty ve scéně. Jejich obalová tělesa jsou vzájemně disjunktní. Lze je také jednodušše automaticky vytvořit z pozice figurky na šachovnici, nebo postupně při definování primitivních objektů. Bylo by možné použít i pravidelné dělení prostoru, neboť rozložení objektů neumožňuje jejich shlukování. Obalová tělesa jsou implementována jako nehiearchická, tzn. pro jeden model je použit jedno obalové těleso. Jako obalové primitivum je použit kvádr rovnoběžný s osami souřadného systému.

3.6 Multiprocessing

Sledování paprsku je velmi náročné na výkon procesoru. Výpočet trvá v rozmezí jednotek sekund až desítek minut či i mnohem déle, v závislosti na složitosti geometrie. Dnes má už i většina běžných pracovních stanic alespoň dvoujádrové procesory. Při klasickém návrhu programu se využije pouze jedno jádro. Metodu sledování paprsku však lze velice snadno paralelizovat. Stačí spustit ten samý program s odpovídajícím parametrem, který specifikuje že se mají počítat jen sudé/liché pixely (v

případě dvou jader), a podobně k-té pixely u více jader. Na konci výpočtu se všechny částečné obrázky spojí do jednoho.

3.7 Modely

Modelem bude nazýván zobrazitelný celek, který se skládá z jednoho nebo více dílčích objektů. Jednotlivými objekty mohou být buďto geometrická primitiva, CSG strom primitiv nebo rotační plocha. Pro každý objekt je definován materiál. Každý model má přiřazeno obalové těleso a vlastní transformační matici pro převod z prostoru scény do prostoru modelu. Program umí následující grafická primitiva:

3.7.1 Rovina

Je definována počátkem a normálou. Pro výpočet průsečíku s paprskem je třeba počátek a normálu převést obecný tvar $a*x + b*y + c*z + d = 0$. Samotný výpočet se provede dosazením parametrického vyjádření paprsku do proměnných x,y,z. Výsledek je parametr t, který lze v případě normalizovaného směrového vektoru považovat za vzdálenost od kamery.

Rovina se při zobrazení výsledné šachové scény nepoužívá, program ji však podoporuje z důvodu odladění chyb a případné flexibilitě při zobrazování jiných scén.

3.7.2 Koule

Je definována počátkem a radiusem. Obecný tvar je $(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2$. Výpočet průsečíku lze opět provést dosazením parametrického vyjádření paprsku. Normála se vypočítá jako rozdíl mezi průsečíkem a středem koule.

3.7.3 Kvádr

Zde je nejdříve paprsek převeden do modelového prostoru. Kvádr se předpokládá, že jeho stěny jsou rovnoběžné s osami souřadného systému. Při výpočtu průsečíku se kontrolují průsečíky s jednotlivými plochami, které se vypočtou dosazením příslušné souřadnice povrchu. Potom už se jenom zkontrolují meze.

3.7.4 Reprezentace figurek

Šachové figurky jsou komplexní objekty. Je složité je věrně namodelovat i pro zkušené grafiky. Většina z nich má podstavu rotačního charakteru a vršek individuálně modelovaný.

První z možností je použít povrchově modelovaná tělesa, které by bylo možné importovat z grafických editorů. Ovšem, pro zajištění hladkých ploch by bylo třeba použít velké množství polygonů. V tomto případě není výhodné používat sledovaní paprsku, ale rasterizaci. Dále je možné

sestavit objekty z primitivních těles (koule, kvádry, kužely, jehlany apod). V tom případě by však bylo třeba modely znatelně zjednoduššit, aby bylo možné objekty zadat ručně. Modelování pomocí implicitních ploch je zajímavá varianta, nicméně je problém vytvořit data, která by mohla popsat dané objekty. NURBS plochy jsou poměrně složité, a proto jsem od nich také upustil, i když by umožnily zobrazit modely v nejlepší kvalitě.

Rozhodl jsem se pro kombinaci CSG a rotačních ploch, které jsou vytvořeny rotací křivky kolem osy.

3.7.5 Rotační plochy

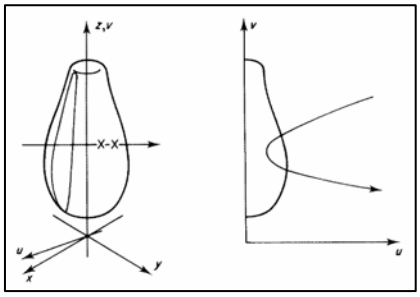
Neboli sweep surfaces, které vzniknou rotací profilu kolem osy. Problém nalezení průsečíku lze zjednoduššit tak, že problém převedeme do 2D. Víme, že pro osu u v prostoru objektu platí $u^2=x^2+y^2$ a pro $v=z$. Z parametrického vyjádření paprsku a těchto dvou vztahů lze odvodit rovnici paprsku ve 2D prostoru: $a*u^2 + b*v^2 + c*v + d = 0$, kde

$a = z^2$

$b = -(x^2 + y^2)$

$c = -2*(-z_0 * (x^2 + y^2) + z*(x_0*x + y_0*y))$

$d = -(z_0^2*(x^2 + y^2) - 2*z*z_0*(x_0*x + y_0*y) + z^2*(x_0^2 + y_0^2))$

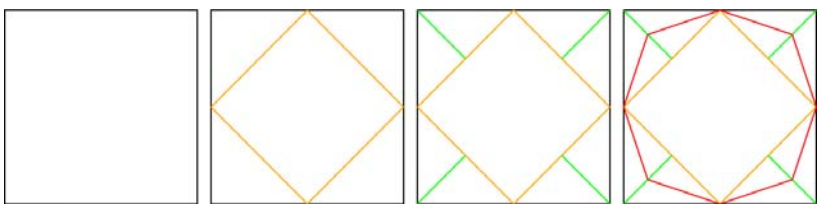


Obrázek 3.2, převzato z [2]

Nyní tedy problém nalezení průsečíku znamená nalézt 2D průsečík křivky paprsku s profilem objektu (viz obr. 3.2), a poté vypočtený průsečík ve uv převést zpět do xyz. Průsečík je vypočítán ve formě vzdálenosti od počátku parametrického vyjádření paprsku.

Profil figurky jsem se rozhodl modelovat pomocí subdivision křivky. S drobnými vylepšeními (viz dále) má podobné vlastnosti jako NURBS křivka. Krom jednoduchosti implementace jsem ji vybral i z důvodu snadnosti výpočtu průsečíku segmentu křivky s křivkou paprsku.

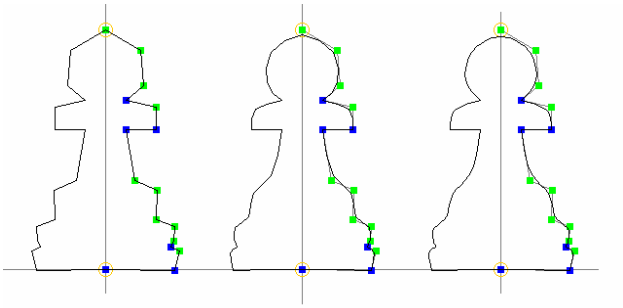
Subdivision, neboli postupně dělené křivky vycházejí ze zadané kostry objektu, což je v podstatě řetěz úseček. Dále lze provést operaci dělení. Zdvojnásobí se počet hran a jejich hrany se zaoblí. Operaci lze dále opakovat a postupně tím získat souvislejší křivku. Existuje více metod pomocí kterých se hrany zaoblují (viz [1]), práce používá následující:



Obrázek 3.3

1. Každá hrana se rozdělí na dvě nové hrany, rozděleny jsou uprostřed.
2. Body hran, které svoje souřadnice měli shodné v přechozím kroku se vypočítají následovně: určí se žlutá pomocná čára, která vznikne spojením středu prvního a druhého segmentu, a ze středu žluté pomocné čáry se vede další zelená pomocná čára do prvního uzlu segmentu. Střed zelené čáry je nová souřadnice pro rozdělenou hranu.
3. Výsledná zaoblená křivka je zvýrazněna červeně.

Seznam segmentů kostry byl ještě rozšířen o příznak pravda/nepravda, který udává zdali se daný bod smí posunout (bod 2). Tím lze dosáhnout ostrých hran i po několika iteracích dělení. V programu potom vypadá dělení profilu následovně (obr. 3.4):



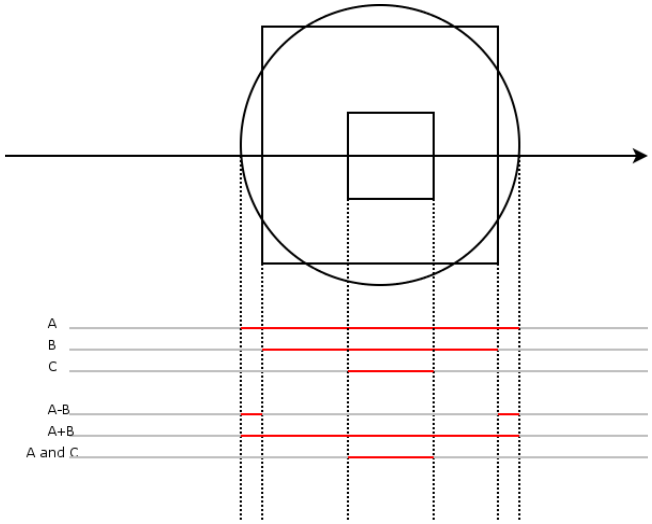
Obrázek 3.4

Pomocí této techniky lze přímo namodelovat pěšáka, střelce, věž a královnu. Výjimka je pro krále, který bude mít na hlavě kříž, a kůň, který bude modelován pomocí CSG. Hodnoty křivek a program pro vizualizaci obr. 3.4. byl převzat z [7].

3.7.6 Konstruktivní geometrie těles

CSG umožňuje pomocí malého množství geometrických primitiv modelovat složitá tělesa. V programu budeme generovat kříž krále a geometrii rytíře. Budeme tedy potřebovat implementovat operace součtu a rozdílu. Zbylý průnik není problém přidat, jak bude zřejmé v dalším textu. Dále je třeba implementovat alespoň grafické primitivum kvádr.

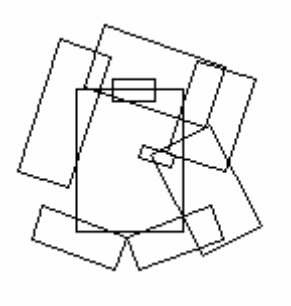
Vyhodnocení CSG stromu lze implementovat následovně, viz obr. 3.5:



Obrázek 3.5

Pro generování jsou např. kresleny tři objekty – dva čtverce a jedna koule. Každý objekt při kolizi s paprskem vygeneruje vlastní přímku (A,B,C), na které umístí průsečíky a označí místo (červeně) kde se nachází v materiálu. Při určování výsledku CSG operace se na výslednou přímku umístí všechny průsečíky ze dvou přímek, a kontrolují se segmenty mezi nimi. Podle logické operace bude buď uvnitř nebo vně materiálu. Průsečíky se na přímku umísťují včetně normály a odkazu na objekt. Přímka se poté stáva podkladem pro další uzel v CSG stromu a pokračuje se od konce stromu ke kořenu. Při vyhodnocování kořene se už najde nejbližší průsečík s materiálem.

Kůň je modelován z kvádru, ze kterého jsou postupně odečítány natočené kvádry různých rozměrů tak aby výsledek byl v rámci možností efektní, viz obr. 3.6.



Obrázek 3.6

3.7.7 Šachovnice

Desku podstavy šachovnice a šachovnici samotnou lze reprezentovat primitivními tělesy typu kvádr. Šachovnici lze popsat jednoduchou procedurální texturou. Nejdříve se převedou souřadnice z prostoru scény do texturového prostoru a poté stačí použít:

```
bool white=true;
if (u%2) white=!white;
if (v%2) white=!white;
```

Dřevo lze generovat pomocí šumu. V jedné dimenzi je mu 20 krát zmenšena frekvence z důvodu imitace pruhů dřeva. Je použit perlinův šum. Implementace je následující:

```
double bumps=perlin(x,y,z*0.05)*20;
double intensity=bumps-((int)bumps);
```

Výsledek sice při menších rozlišeních poněkud trpí aliasingem, nicméně při použití supersamplingu je výsledek akceptovatelný.

3.7.8 Hodiny

Jsou nejjednodušším objektem ve scéně. Jsou namodelovány jako kvádr. Ciferník s ručičkami zobrazí namapovaná bitmapová textura.

3.8 Materiály

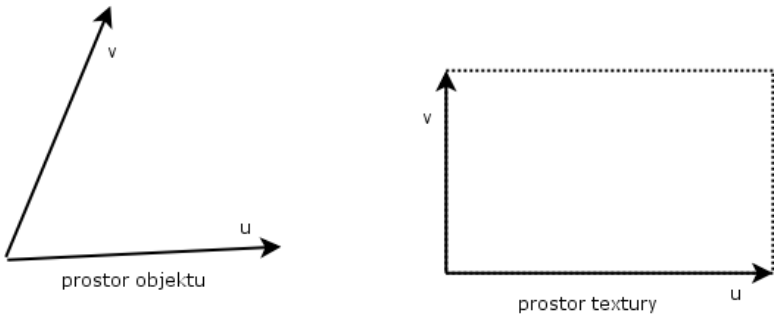
Materiálem je myšlen souhrn konstant Phongova osvětlovacího modelu popsaného v kapitole 2.3.1, který definuje povrch. Každý objekt uvnitř modelu má přiřazen vlastní materiál. Při výpočtu průsečíků je vrácena vzdálenost a definice materiálu. Ta se může zkopírovat buď z přiřazeného materiálu, nebo z funkce provádějící mapování textury.

3.8.1 Mapování textur

Program podporuje mapování jak bitmapových, tak i procedurálních textur. U bitmapových v současnosti umožňuje pouze mapování textur na planární plošinu. Pro cylindrické a kulové mapování je však připraven prostor a proto není problém je do programu doplnit.

3.8.1.1 Planární mapování

Pro každou texturu je definovaný mapovací prostor, který se skládá ze souřadnice počátku prostoru a vektorů *u* a *v*, které značí osy souřadného systému. Velikosti vektoru odpovídají velikosti textury. Směr vektorů určí namapování prostoru textury. Tím lze kontrolovat plochu a zkosení namapované textury na objektu (obr 3.7). Pokud plocha objektu přesahuje definované vektory textury, textura se bude opakovat.



Obrázek 3.7

Nejdříve se bod v prostoru převede na souřadnice v *uv* prostoru textury. Bod v prostoru je tedy lineární kombinací vektorů *u* a *v*. Tedy např. $p=u*x + v*y$, pokud bychom mapovali na *xy* plochu. Celou rovnici lze vyjádřit jako matici a výsledné *u* a *v* souřadnice v prostoru textury vypočítat kramerovým pravidlem.

Materiál se z bitmapy vypočítá tak, že je difuzní barva nahrazena barvou z textury. Mapování bitmap se ve scéně používá jenom pro hodiny.

Planární mapovaní je ještě použito u výpočtu procedurální textury pro šachovnici. Po výpočtu uv souřadnic v xz prostoru je již jenom proveden výpočet uvedený v 3.7.7.

3.9 Program

Jedno z možných využití je, že program bude vykreslovat stav šachovnice nějakému šachovému programu. Je tedy třeba vyřešit předávání parametrů. Protože program nepotřebuje předpočítávat data či načítat rozsáhlé scény, je akceptovatelné spouštět ho pro každý výpočet zvlášť. Parametry se tedy budou předávat přes příkazovou řádku. Zadán bude výstupní soubor a pozice figurek na šachovnici. Pozice budou zadány dvěmi řetězci v následujícím formátu: $N_1x_1y_1N_2x2y_2...$, kde N_x je písmeno které značí figurku a x_n=<A,H> a y_n=<1,8> pozici na šachovnici. Figurky mají následující písmena:

- K – král
- D – dáma
- V – věž
- S – střelec
- J – jezdec
- P – pěšec

Obrázek 3.8, [8]

Například pro výchozí rozestavení bílého hráče to bude:

VA1JB1SC1KD1DE1SF1JG1VH1PA2PB2PC2PD2PE2PF2PG2PH2.

4 Implementace

Při návrhu programu byl problém implementace rozdělen do více sekcí. Tyto sekce jsou na druhých více nebo méně závislé. Vzhledem k tomuto faktu jsem se rozhodl program implementovat v objektově orientovaném C++. Abychom mohli začít s popisem hlavních částí programu, je třeba nejdříve implementovat pomocné třídy.

4.1 Pomocné třídy

4.1.1 Vektorová algebra

Raytracing je postaven na vektorové algebře. Je tedy výhodné ušetřit si práci a definovat si datový typ Vektor a běžně používané operace implementovat pomocí přetěžování operátorů. Tato třída *Vector* je uložena ve zdrojových souborech Vector.cpp a Vector.h. Implementuje následující operace:

- normalizaci vektoru
- součet, rozdíl, součin s jiným vektorem
- součin s konstantou
- skalární a vektorový součin s vektorem
- délku vektoru
- výpočet úhlu s jiným vektorem
- vynásobení s transformační maticí
- vytisknutí obsahu do konzole

Současně zdrojové soubory obsahují definici paprsku *Ray*, kterou tvoří dva vektory – jeden použit pro počátek a druhý pro směr.

4.1.2 Maticová algebra

Pro výpočet transformací je výhodné a přehledné pracovat s homogenními souřadnicemi. Pro tyto účely je zhotovena pomocná třída Matrix pro práci se 4x4 maticemi. Je v souborech Matrix.cpp a Matrix.h. Implementuje následující operace, na základě popisu v kap. 3.2.:

- nastavit jednotkovou matici
- nastavit matici konstantami posunutí, změny měřítka a rotace
- vynásobit matici maticí posunutí, změny měřítka, rotace pro x,y a z nebo pro všechny osy zároveň.

- násobení s jinou maticí
- výpočet inverze a transponované matice
- vytisknutí obsahu do konzole

4.1.3 Třída pro zpracování obrázků

Pro vstup a výstup bitmap raytraceru byl zvolen souborový formát BMP. Obraz je uchováván ve třídě Image, která je v souborech Image.cpp a Image.h. Obsahuje

- funkce pro alokaci paměti pro obraz Initialize(width,height)
- vyplnění obrázku zvolenou barvou Fill(Pixel color)
- funkce pro získání a uložení pixelu SetPixel(x,y,color) a Pixel GetPixel(x,y).
- a funkce pro vstup a výstup obrázku LoadBMP(name) a SaveBMP(name).

Pro práci s pixely je vytvořena třída Pixel. Je ve stejném souboru jako Image. Obsahuje funkci FromVector() pro konverzi dat z proměnné typu Vector a naopak (ToVector()). Transformuje barevný prostor s maximálními hodnotami (1,1,1) do RGB (255,255,255). Implementace je založena na [9].

4.2 Objekty

S ohledem na navrženou scénu jsem se rozhodl pro následující terminologii:

Modelem bude nazváno komplexní těleso, které je vytvořeno z grafických primitiv a/nebo CSG a/nebo rotačních ploch. Příkladem modelu budiž šachová figurka. Každý model má svoje obalové těleso, vlastní transformační matici a seznam objektů. Model je reprezentován třídou *Model*.

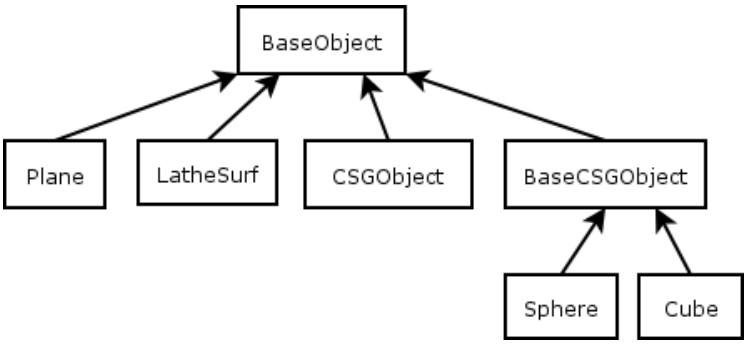
Objekty jsou jednotlivá primitiva. Za objekt je považován buď primitivní těleso, strom CSG objektů nebo definice rotační plochy. Každý objekt má definovanou svoji pozici vzhledem k transformační matici modelu. Dále obsahuje definici materiálu pro osvětlení a odkaz na příslušnou texturu.

Třída *Model* obsahuje metodu *Intersect*(Ray), která nejdříve převede paprsek do lokálního modelového prostoru. Poté provede test jestli paprsek protíná obalové těleso, a pokud ano tak prochází všechny vnořené objekty a hledá s nimi průsečíky. Vrací nejbližší průsečík. Průsečík se vrací ve formě datového typu (třídy) *Intersection*, který obsahuje konstantu vzdálenosti průsečíku (parametr *t* parametrického vyjádření normalizovaného paprsku), paprsek s kterým průsečík nastal, odkaz na objekt kolize a normálu v bodě průsečíku.

Základem pro všechny objekty je třída *BaseObject*. Ta obsahuje definici materiálu, odkaz na případnou texturu, typ objektu, transformační matice a odkaz na model, který objekt vlastní. Podobně jako *Model* obsahuje *BaseObject* metody pro převod z modelového prostoru do prostoru objektu. Materiál je zpřístupněn přes metodu *Material GetMat(Vector pos)*, který buď zkopíruje phongovu definici materiálu, anebo v případě specifikace textury vygeneruje potřebnou definici. Dále obsahuje virtuální metodu *Intersection Intersect(Ray)*, která je implementována různě v děděných třídach. Vrací nejbližší průsečík s paprskem, pokud nastane. Pokud ne tak nastaví odkaz na objekt v typu *Intersection* na *NULL*. Poslední virtuální metoda *bool Inside(Vector)* vrací *true* nebo *false* v případě že daný bod leží uvnitř nebo vně objektu. Tato metoda je implementována jenom jako rozšíření pro případ, že by někdo chtěl modelovat průhledné objekty. Při refrakci je třeba vypočítat snellův koeficient lomu, který je závislý na pořadí hustot materiálů přes něž paprsek prochází.

Třídu *BaseObject* přímo dědí grafické primitivum plocha(třída *Plane*), *CSGObject a LatheSurf* (třída pro rotační plochy). Ostatní primitiva jsou děděna přes třídu *BaseCSGObject*, která dědí od *BaseObject*. *BaseCSGObject* rozšiřuje svého rodiče o virtuální metodu *CSGIntersectList IntersectCSG(Ray)*, která vypočítá přímku všech průsečíku paprsku s objektem způsobem uvedeným v kap. 3.7.6. Z této přímky potom může virtuální metoda *Intersect* přímo určit nejbližší průsečík, pokud by byl objekt používán jako primitivum. V případě výpočtu CSG objektu se základem stává *CSGObject*. Tato třída obsahuje kořen CSG stromu, který reprezentuje typ *CSGNode*. CSGNode obsahuje definici CSG operace, která může být sjednocení, průnik a rozdíl. Dále obsahuje levé a pravé kořeny. Ty mohou buďto být odkaz na *BaseCSGObject*, nebo *CSGNode*. Metoda třídy *CSGObject*, *IntersectCSG(Ray, CSGNode)* je poté rekurzivně použita pro postupné generování přímky průsečíků *CSGIntersectList*, tzn. Postupně se volají IntersectCSG a výsledné seznamy se kombinují metodou *CombineCSGLists(operation, leftlist, rightlist)*. Nakonec se z v metodě *Intersect* třídy *CSGObject* určí nejbližší průsečík který je výsledkem operace.

Graf dědičnosti ukazuje obrázek 4.1:

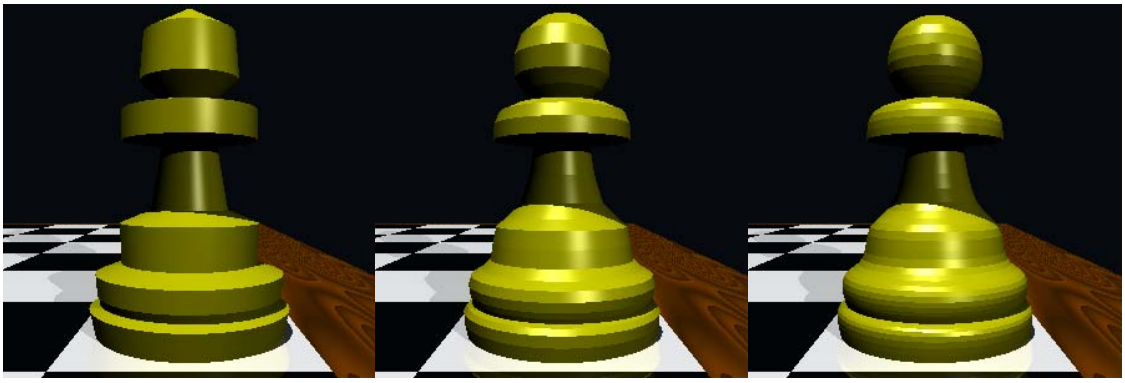


Obrázek 4.1

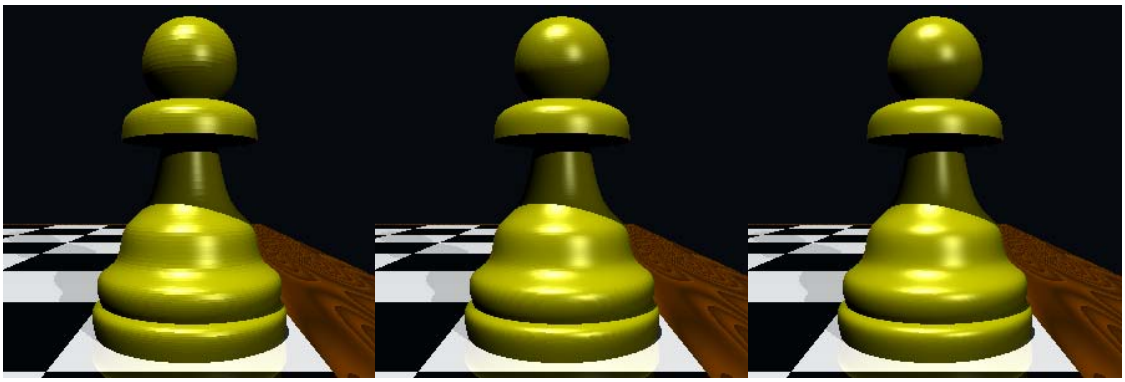
Uvedené třídy, včetně výpočtů jsou definované a deklarované v souborech *Model.cpp* a *Model.h*, s vyjímkou rotačních ploch a výpočtu textur.

Třída *LatheSurf* sloužící k výpočtu rotačních ploch používá třídu *SubdivisionCurve*, umístěnou v souboru *Curve.cpp* a *Curve.h*. V tomto souboru jsou také umístěny další funkce pro práci s křivkami. Při výpočtu Intersect se nejdříve stejně jako u ostatních objektů nejdříve převedou souřadnice paprsku z modelového prostoru do prostoru objektu. Poté se převede paprsek do *uv* prostoru křivky a vypočítají koeficienty *a,b,c,d* obecného vyjádření křivky. Poté se postupně hledají průsečíky s jednotlivými segmenty (linkami) subdivision křivky. Výpočet je metoda *SegmentIntersectCurve* třídy *SubdivisionCurve*. Parametrem je index segmentu a koeficienty křivky paprsku. Vrací typ *IntersectInfo*, což je obdoba typu *Intersection* aplikovaná do prostoru křivky. Průsečík se vypočítá opět dosazením parametrického vyjádření segmentu do obecného vyjádření křivky. Jedná se o kvadratickou rovnici, kterou řeší pomocná třída *QuadraticSolver*. Výsledný průsečík je *u,v* souřadnice, včetně normály v *uv* prostoru. Normála se určí metodou SegmentGetNormal jako vektor kolmý na segment křivky a znaménko vektoru podle kvadrantu kam segment směřuje. Vypočítaný *IntersectInfo* se pomocí metody *GetDistFromUV(Ray,u,v)* se převede na vzdálenost t (parametr) paprsku. Výpočet lze opět odvodit z podmínek $x^2+y^2=u^2$, $v=z$ a parametrického vyjádření paprsku. Podobně se přepočítá normála. Ta se nakonec převede z prostoru objektu do prostoru světa a přidá do typu *Intersection*.

Jak bylo uvedeno v kapitole 3.7.5, subdivision křivka je zadána strohou kostrou, která se vyhlazuje opakovaným dělením. Toto dělení zajišťuje metoda *Subdivide* třídy *SubdivisionCurve*. Pro hladký vzhled výsledných ploch je třeba použít více dělení. Jejich počet závisí na rozlišení výsledného obrazu, nasvícení světly, barvě materiálu a i dojmu pozorovatele. Je proto určen experimentálně, hodnota je nastavena na 4, která by měla postačovat. Vzhled po jednotlivých dělení je postupně zobrazen na následujícím obrázcích:



Počet dělení: 0 1 2



Počet dělení: 3 4 5

Doba výpočtu 300x300 obrázku na 2.7 ghz jádře byla postupně: 2.9s, 3.2s, 3.8s, 5.4s, 7.8s a 13.1s. Jak je vidět, zdojnásobení hran vede ke složitosti $T(n)=2^n$.

4.2.1 Výpočet textur

Jak už bylo popsáno v předchozí kapitole, metoda *GetMat* třídy *BaseObject* vrátí definovaný materiál, a v případě specifikované textury (ukazatel na texturu není NULL) zavolá metodu *GetMat(Vector)* textury.

Všechny textury dědí z rodičovské třídy *Texture*. Ta obsahuje počátek a vektory *u,v* pro nastavení parametrů mapování. Dále typ, který rozlišuje typ mapování 2D textury, který může být *LINEAR*, *CYLINDRICAL* a *SPHERICAL*. Obsahuje metodu *ComputeUV(Vector)*, která na základě typu mapování vypočítá uv souřadnice v prostoru textury podle výpočtu uvedeného v kapitole 3.8.1. Implemetuje pouze planární (linear) mapování, neboť jiný typ se v šachové scéně nevyskytuje. Tato metoda je potom použita v synovských třídách výpočtu textury. Všechny třídy textur jsou umístěné v souborech *texture.cpp* a *texture.h*.

Obor bitmapových textur reprezentuje třída *BitmapTexture*. Obsahuje metodu *bool LoadTexture(path)*, která načte BMP soubor. Metoda *GetMat* převede souřadnici bodu do uv prostoru textury, vytvoří materiál s výchozími hodnotami a nahradí difuzní složku barvy barvou z textury.

Procedurální texturu šachovnice tvoří třída *CheckerTexture* a implementuje ji podobně jako bitmapovou – převede souřadnice do prostoru textury a místo načtení z bitmapy jednodušše vypočítá barvu tak jak bylo demonstrováno v kapitole 3.7.7.

Procedurální texturu dřeva tvoří třída *WoodTexture*. Je založená na výpočtu perlinova šumu. Implementace samotného výpočtu šumu byla převzata z [3], a je v souborech *perlin.c* a *perlin.h*, proto ji nebudu popisovat. Je vypočtena intenzita textury podle algoritmu uvedeného v kapitole 3.7.7 a nastavena difuzní barva na hnědou a difuzní koeficient na intenzitu textury.

4.3 Kamera

Přidáním nastavitelné kamery umožníme snadnější a inuitivnější nastavení pozice pozorovatele v scéně. Třída *Camera* je ve zdrojových souborech *Camera.cpp* a *Camera.h*. Obsahuje následující metody:

- nastavení pozice kamery *SetPosition(Vector v)*
- nastavení směru pohledu *LookAt(Vector v)*
- funkce pro převod ze souřadnicového prostoru světa do prostoru kamery a naopak
- funkci *Ray GetRay(x,y,screenWidth,screenHeight)*, která vypočítá vržený paprsek z kamery do scény. Funkce počítá s nastavitelnými parametry zorného úhlu FOV, poměrem stran obrázku a natočením kamery podle osy z. Implementuje algoritmy uvedené v 3.1.

4.4 Raytracer

Třída *Raytracer* v souborech *Raytracer.cpp* a *Raytracer.h* obsahuje seznamy modelů a světel, kameru, výstupní obrázek, parametry prostředí a jiné. Obsahuje metodu Render(), která prochází jednotlivé pixely obrázku. Pokud byly nastaveny parametry pro multiprocessing, pak vynechává z výpočtu ty pixely, jejichž zbytek po dělení počtem procesoru se nerovná zadanému pořadovému číslu programu (v programu proměnná offset). Pro pixely vypočítává barvu pomocí funkce

Trace(ray, maxrekurzi, bool StinovyPaprsek, maxhloubka).

Ray je paprsek, který se získá voláním metody *GetRay* objektu *Camera*. Booleovská hodnota *StinovyPaprsek* určuje zdali se jedná o stínový paprsek. Maxrekurzí značí maximální úroveň rekurze. Při překročení této úrovně vrátí vektor (0,0,0) a výpočet již nepokračuje dále. Maxhloubka značí nejdelší vzdálenost od kamery, ve které se zjišťují průsečíky. Využije se pro výpočet stínového paprsku, kdy všechny uvažované průsečíky musí být v menší vzdálenosti než je světlo. Následuje popis činnosti funkce.

- Nejdříve se vyhledá nejbližší průsečík paprsku s objekty
- Pokud se jedná o stínový paprsek a nebyl nalezen žádný průsečík, pak vrátí barvu (0,0,0), jinak (1,1,1)
- Pokud nebyl nalezen žádný průsečík, pak je výsledkem barva pozadí.
- Jinak se pokračuje podle metody phongova osvětlení: pro všechny světla se spočítá stínový paprsek, difuzní osvětlení, příp. spekulární odlesky.

- pokud má materiál nastavenou odrazivost nebo průhlednost, pak se vyšlou rekurzivně další paprsky.
- barva od phongova modelu, odrazová a průsvitná barva se sečte a určí tak výslednou barvu pixelu.

4.5 Ostatní

Dále program obsahuje další třídy, jako světlo, paprsek apod. Jsou to však více datové typy jak programové celky. Proto budu předpokládat že jejich význam bude zřejmý z popisu programu.

Program podporuje využití více jader. V hlavní funkci *main* v souboru *main.cpp* nejdříve program přečte parametry. Pokud najde parametr s počtem CPU větší než dva, spustí N dalších programů, a předá jim stejné parametry. Navíc k nim přidá parametr *–cpuoffset N*, který značí že se jedná o synovský proces a že se mají počítat pouze pixely, jejichž souřadnice *x* splňuje podmínku *(x mod cpu_number)=cpuoffset*.

4.6 Vstup dat

Styl předávání parametrů odráží uvažované využití programu jako vykreslovací část šachového stroje či demonstrační program. Proto se nepředpokládá, že by se měnila geometrie nebo objekty scény. Mění se pouze pozice figurek, případně pozice kamery nebo nasvícení scény.

4.6.1 Geometrie

Definice geometrie objektů, jejich textur a pozice jsou proto umístěny v programu. Metoda *Make(object, parametr)* třídy *Model* umožňuje aktuálnímu modelu přiřadit požadovaný vzhled. Jako parametr *object* lze zadat šachové figurky (*OBJ_PAWN, OBJ_KNIGHT*, …) a pomocné objekty jako šachovnice, hodiny a prostředí. Definice dat křivek jsou umístěné v souboru ObjectDefs.cpp, ostatní jsou přímo v těle funkce Make, včetně vytvoření potřebných materiálů a textur, CSG stromů, nastavení obalových těles apod.

4.6.2 Nastavení scény

Nastavení pozice a směru kamery, světla, rozlišení výstupu a další jsou načítány ze souboru *config.ini*. Je v běžně používáném ini formátu systému Windows:

```
[GlobalSettings]
ImageWidth=300
ImageHeight=300
BackgroundColor=0,0,0
LightCount=2
ModelCount=0
SnellIndex=1.0
```

Názvy proměnných jsou samopopisující. *SnellIndex* je index lomu pro průhledné materiály. V současnosti program podporuje pouze jeden průhledný materiál. *Lightcount* je počet sekcí definujících světla. Sekce jsou pojmenována jako *Light*<index světla>. Příklad definice:

```
[Light0]
Position=0,150,-50
Color=0.8,0.8,0.8
```

Kamera má vlastní sekci Camera. Je zadána pozice, bod pohledu, zorný úhel a rotace v úhlu pohledu.

```
[Camera]
Position=45,15,-30
LookAt=14,0,14
FOV=90
ZRot=10
```

4.6.3 Příkazová řádka

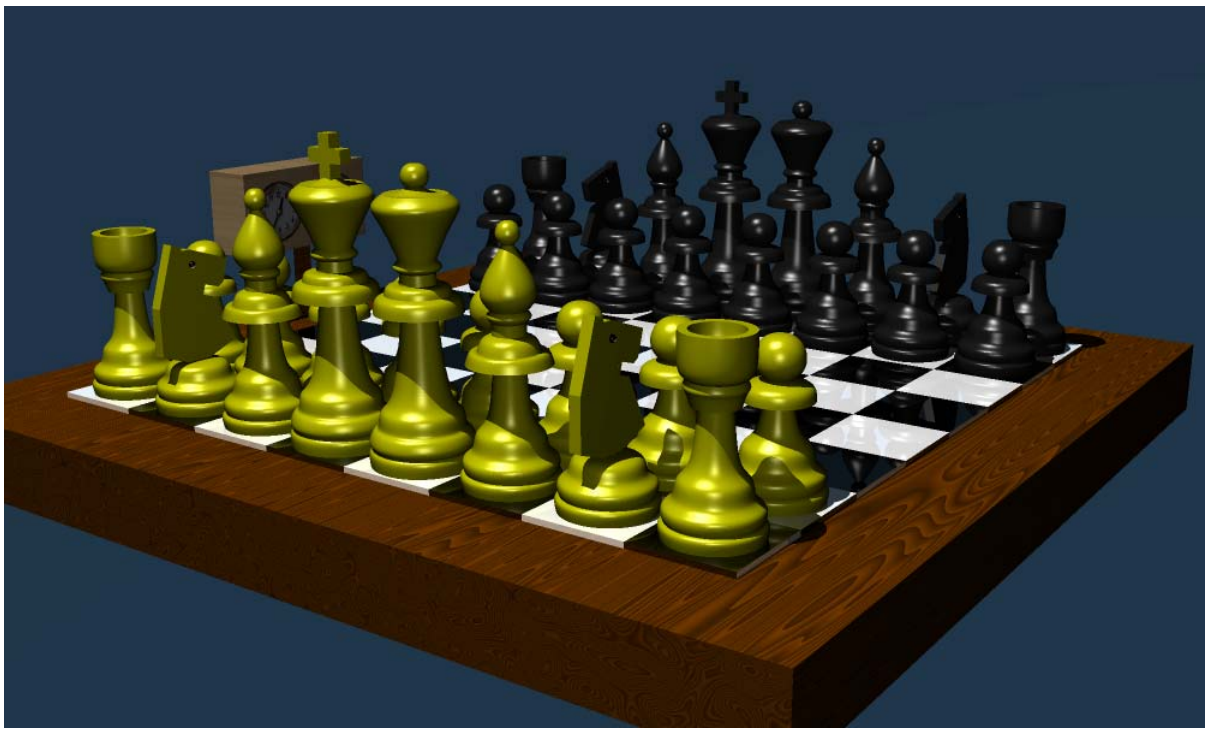
Program očekává zadání takových parametrů, které se mohou lišit obrázek od obrázku, anebo jsou závislé na hardwaru. Použití je: *jméno_programu –variable1 value1 –variable2 value2* atd:

Proměnná a parametr	*význam*
-config <soubor.ini>	značí konfigurační soubor ve formátu popsaném v kapitole 4.6.2
-output <soubor.bmp>	jméno výstupního obrázku
-AA <0,1,2>	značí režim antialiasingu. 0 je pro žádný, 1 je supersampling, 2 je adaptivní supersampling. Adaptivní není možné použít v případě multiprocessingu.
-ssrays <1,2,3,…>	počet paprsků použitých pro supersampling. Číslo značí pouze počet pro horizontální osu, celkový počet paprsků je N^2. Pro kvalitní výstup stačí použít nejvýše hodnotu 3.

-cpu <2,3,…> -whiteplayer <řetězec>	počet procesů které program celkem vytvoří pro multiprocessing řetězec pozic bílých figurek popsaný v kapitole 3.9
-blackplayer <řetězec>	to samé pro černé figurky

5 Zhodnocení a závěr

Podařilo se splnit zadání a zobrazit šachovou partii pomocí metody sledování paprsku. Byla navržena a implementována ne zcela běžná metoda zobrazování rotačních ploch a potřebný rozsáhlý zbytek vykreslovacího programu. Příklad výstupu je na následujícím obrázku:



Obrázek 5.1

Výstup lze již s větší mírou nadsázky označit jako realistický. Problémem je však doba výpočtu. Výše zobrazený obrázek (1000x600, 4 paprsky/pixel antialiasing) program počítal 3.8 minut na dvoujádrovém AMD X2 5000+ (2 x 2.7 Ghz). To je zcela jistě neúnosné pro použití i v šachovém stroji, kde nároky na odezvu nejsou příliš vysoké. Urychlit lze na úkor kvality obrázku. Při vypnutí antialiasingu výpočet trval jen 56 sekund. Při snížení úrovně subdivision křivky ze 4 na 2 to bylo už 31 sekund při akceptovatelném vzhledu. Dalším snižovaním detailů (třeba vypnutí odrazivosti šachovnice, …) či rozlišení by se dalo dostat i na nižší čas v řádu jednotek sekund.

Program by se však dal dále urychlit i optimalizací. Samotná režie výpočtu paprsků lze odstranit zavedením hiearchie obalových těles nebo je zkombinovat s metodami dělení prostoru. Další úzké hrdo je v detekci průsečíků křivky paprsku se segmenty křivky. Zde by bylo vhodné aplikovat některou z metod dělení prostoru.

Stávající návrh lze jednodušše rozšiřovat – objektový návrh k tomu přímo vybízí. Využití by se našlo v akademické oblasti, ať už pro demonstraci technik sledování paprsku, nebo rozšiřování dalšími možnostmi. Lze zkoumat techniku subdivision křivek, které se běžně příliš nepoužívají. Program se dá i s drobnými omezeními v kvalitě používat jako vykreslování šachového stroje. Anebo zkoumat a vyvíjet možnosti multiprocessingu, případně rozšířit program o distribuovaný výpočet.

Anebo, pokud by se nehodil ani na jedno z předchozího, lze použít jako zajímavý benchmark.

Literatura

[1] Žára, J., Beneš, B., Sochor, J., Felkel, P. *Moderní počítačová grafika*. Computer Press 2004.

[2] Glassner A. S. *An introduction to ray tracing*. Academic Press 1991, ISBN 0-12-286160-4

[3] Perlinův šum, vč. zdrojových kódů, dokument dostupný na URL: http://ozviz.wasp.uwa.edu.au/~pbourke/texture_colour/perlin/

[4] Phongův osvětlovací model, dokument dostupný na URL: http://en.wikipedia.org/wiki/Phong_shading

[5] Slajdy předmětu PGR na FIT VUT, 2007

[6] Slajdy předmětu IZG na FIT VUT, 2007

[7] Teorie subdivision křivek a demonstrační programy, URL: http://www.subdivision.org/demos/

[8] Obecné informace o hře šachy, dokument dostupný na URL: http://en.wikipedia.org/wiki/Chess

[9] Zdrojové kódy cvičení z PGR a MUL na FIT VUT, 2007

Seznam příloh

Příloha 1. CD

Příloha 1. CD

Součástí diplomové práce je CD, které obsahuje následující adresáře:

/pictures – ukázkové obrázky

/RayTracer

- /src – zdrojové kódy aplikace
- /debug
- /release – tyto dva adresáře obsahují spustitelný program, včetně textury a konfiguračních souborů.
- V adresáři jsou projektové soubory pro Visual Studio 2005.

/text – text této práce ve formátu DOC a PDF.